Novel Techniques for Processing Unstructured Data Sets

Roger Chamberlain
Ron K. Cytron

Novel Techniques for Processing Unstructured Data Sets

Roger D. Chamberlain and Ron K. Cytron
Data Search Systems, Inc.
4041 Forest Park Ave.
Saint Louis, Missouri 63108
roger@dssimail.com, rcytron@dssimail.com

Abstract—While improvements in the density of semiconductor circuitry have been dramatic, the density improvements in magnetic storage have been even greater. We now store much more data than we have time to process, implying that techniques for processing these data need to be significantly altered. This paper describes a new architectural approach that enables the processing of very large data sets, yielding two orders of magnitude performance gain over conventional approaches.[1,2]

TABLE OF CONTENTS

1. INTRODUCTION

The remarkable improvements in magnetic storage density over the past several years have outstripped the improvements in semiconductor technology and associated improvements in processor performance. At the same time interconnect and bus technology performance gains have not kept up with either. The result is dramatically lower prices for magnetic storage and an inability to keep up with the computing and bandwidth requirements for serious analysis of large data sets. This performance gap is particularly acute for unstructured data. We address this fundamental and growing gap between data storage capabilities and computing requirements with a new system architecture and application implementation paradigm.

At its core, our approach removes burdensome high-volume data processing tasks from the assigned workload of general-purpose processors and: 1) executes processing tasks in reconfigurable hardware rather than in software, thereby significantly gaining processing power; 2) replicates the reconfigurable hardware across the data store, thereby taking advantage of process parallelism to reduce processing times and allow data storage to grow unbounded without decreasing performance; and 3) positions the hardware close to the data, thereby reducing processing delays associated with contention for interconnect and bus system components. The result is a system that is capable of significant (orders of magnitude) performance gains over that of conventional systems.

The reconfigurable hardware consists of field-programmable gate arrays (FPGAs) that are programmed (configured) using firmware. We have developed a standard interface that communicates with the system and applications using standard drivers and APIs, respectively. The result is a readily modifiable infrastructure that supports a wide class of application functionality.

In our previous work, we have demonstrated high throughput I/O performance from the data store to the FPGA as well as a set of applications that includes exact text search, approximate text search, biosequence similarity search, etc. [1,2,3,4,5] Measured performance gains for these applications range from one to two orders of magnitude over state-of-the-art commodity processors.

In this paper, we will compare and contrast our techniques with commonly used approaches for extracting information from large, unstructured data stores. We will also describe the process by which an application can be deployed on our system. A major task in this process is the decomposition of applications into firmware portions that are executed on reconfigurable hardware and software portions that are executed on general-purpose processors.

The goal is to enhance the fundamental ability of users to draw useful conclusions from huge volumes of otherwise uninformative data. This requires dramatic performance gains in how we look through the raw data and extract potential items of interest.

2. TECHNOLOGY TRENDS

Exponential Growth

Virtually everyone connected with the electronics industry is familiar with Moore's Law. In 1965, Gordon Moore observed that component densities on silicon integrated circuits tended to double each year [6]. While the rate has slowed somewhat in the subsequent four decades, the exponential growth trend has continued unabated.

An equally impressive, yet less well publicized, trend has been the increase in areal density of magnetic storage media. Figure 1 (from [5]) plots the storage density achievable in

[1] 0-7803-8870-4/05/$20.00© 2005 IEEE
[2] IEEEAC paper #1388, Version 5, Updated December 10, 2004

both semiconductors (as measured by commercial DRAM) and magnetic disks over the past several decades. Over the entire period, the growth rate in semiconductor density has been relatively constant. Until the early 1990's, the growth rate in magnetic bit density was lower than that of semiconductors, bringing the two curves closer to one another. Since that time, however, the growth rate in magnetic bit density has eclipsed that of semiconductors, and the unit cost of disk storage has experienced a commensurate decline.

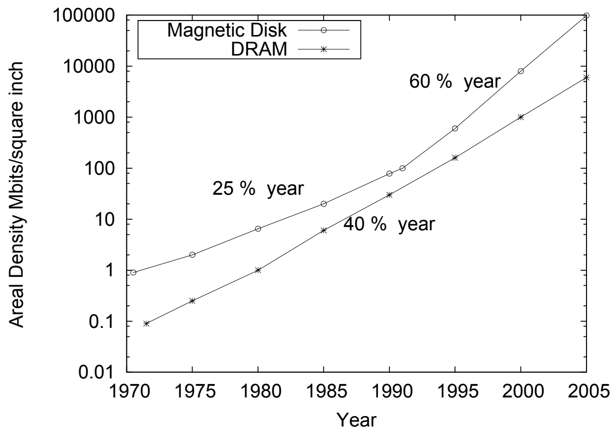

Figure 1. Semiconductor and magnetic bit density [5].

To further complicate the situation, interconnect bandwidths haven't kept pace with either of the above trends. Figure 2 (also from [5]) plots a best fit line associated with typical bus system bandwidths (e.g., PC, XT, ISA, PCI, PCI-X, PCI-Express, Fibre Channel, IDE/Ultra ATA, SATA, etc.). The growth rate in bandwidth is significantly lower than that of either semiconductor or disk densities.

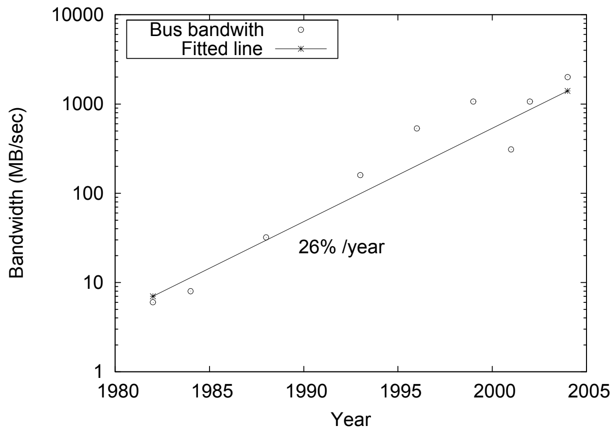

Figure 2. Disk bus bandwidth [5].

One implication of the above trends is that I/O performance is more frequently a limiting factor in the usefulness of systems, especially when the application of interest relies on large data volumes. This is particularly true for unstructured data sets, where it is typically necessary to access the bulk of the data to perform the requested computation.

Reconfigurable Hardware

A second technology trend is the recent availability of affordable reconfigurable hardware with significant computational capacity. In the last decade, field-programmable gate arrays (FPGAs) have evolved away from small, expensive devices that were limited in their applicability into realistic design choices for high-volume, production systems. Current capacities exceed 8 million gate equivalents, and the incremental costs to include them in the system are quite reasonable. Significant research advances have been made in the area of reconfigurable computing (see [7] and [8]) and commercial offerings are available from both Altera [9] and Xilinx [10].

A significant fraction of the current production of FPGAs is used as replacements for application-specific integrated circuits (ASICs), supporting functionality changes when a new application version is released. We, however, are explicitly taking advantage of the ability to frequently reconfigure these devices. Instead of simply loading one configuration into the part for the duration of the system execution, we optionally reconfigure the part for each application. With reconfiguration times measured in 10s of milliseconds [11], an application context switch does have a higher time cost than is typical with a conventional processor, but it is quite reasonable, none the less.

3. ARCHITECTURE

Our novel approach for processing of unstructured data sets exploits both the computing capabilities of reconfigurable hardware to accelerate processing and moves the computation closer to the data, thereby reducing the need to transport the data across multiple busses and interfaces prior to the actual processing taking place.

The overall system architecture is illustrated in Figure 3. A set of compute nodes are interconnected via a network, and the disk drives that store the data set are partitioned across the compute nodes. Each compute node contains one or more traditional processors, memory, one or more FPGAs, disk controllers, and the network interface that provides connectivity to the other compute nodes.

The FPGA positioning and nominal high-volume data flow in our system are illustrated in Figure 4. Data flows off the disk subsystem into the FPGA. The FPGA provides reconfigurable logic that has its function specified via firmware. We have designed and built a standard firmware socket that handles data movement requirements into and out of the FPGA, providing a consistent application interface to firmware application modules that are deployed within the FPGA. Results of the processing performed on the FPGA are delivered to the processor. By delivering the high-volume data directly to the FPGA, the processor can be relieved of the requirement of handling the bulk of the original data set.

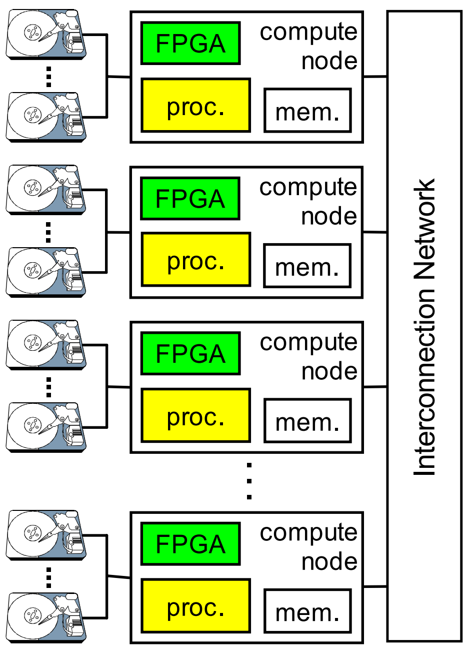

Figure 3. System architecture.

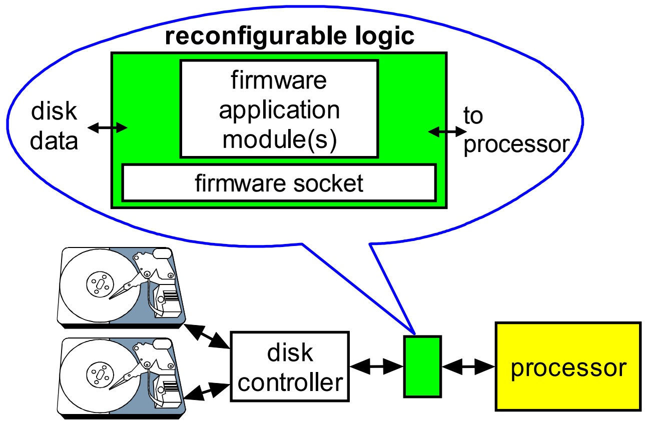

Figure 4. Compute node architecture.

4. Application Deployment

Figure 5 shows the framework for the deployment of applications on our system. The top three layers represent functionality that is executed in software on the compute node's processor, while the bottom two layers represent functionality that is executed in firmware on the FPGA.

The central three layers of this framework are fairly application independent, supporting data movement between the application software at the top of the framework and the firmware application module(s) at the bottom of the framework.

Another way to view the application decomposition is shown in Figure 6. The top portion of the pyramid represents application functionality performed in software, and the bottom portion of the pyramid represents application functionality deployed in firmware. The firmware, with its greater execution speed, is well-matched to application functions that need to examine a greater data volume. The software, with its ease of programmability, is well suited to application functions that have greater algorithmic complexity.

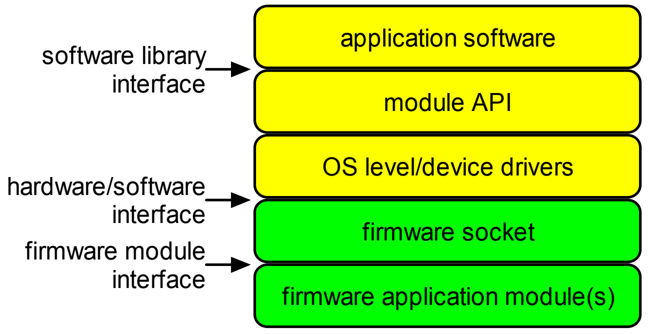
software library interface
hardware/software interface
firmware module interface
application software
module API
OS level/device drivers
firmware socket
firmware application module(s)

Figure 5. Application deployment framework.

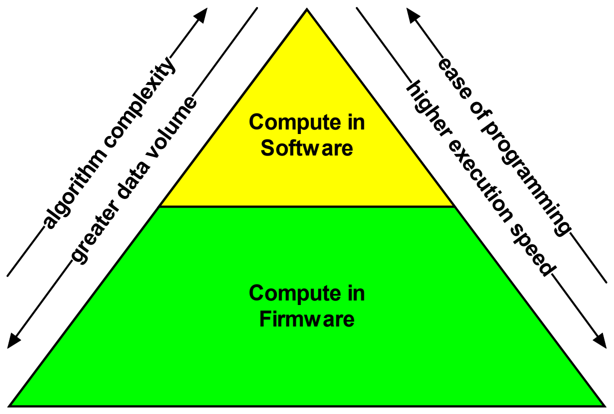

Figure 6. Application deployment pyramid.

We illustrate this decomposition using the example of an approximate search application. Consider a search (within a set of unstructured text files) for matches on the following query:

(Czar AND Nicholas) NEAR Crimea

In addition, it can be specified that the first word may have up to two characters different than the literal string (so as to enable matching the text "Tsar" as well as "Czar") and that the NEAR operator is parameterized to require a match within 1000 characters (i.e., its arguments must reside within 1000 characters of one another).

In our current implementation of this approximate search application, the individual keywords in the query (i.e., "Czar," "Nicholas," and "Crimea") are found through firmware (including the approximate matches) and the logical combinations of these results (i.e., the AND and NEAR operators) are computed in software. In this way the firmware considers the bulk of the raw data, and the software operates on results returned by the firmware.

Next we consider the more general application decomposition required in a data mining operation. The firmware is assigned responsibility for the simple, high data-rate, repetitive operations that must examine the entire data set, effectively filtering the raw data into a more manageable size prior to the higher-level processing. The software is assigned responsibility for processing the filtered output looking for high-level semantic information. This can include probability modeling, knowledge-base modeling, etc.

Given a desired application decomposition, the actual deployment of the application requires both the software and the firmware to be developed. Given the complexity of firmware development, we anticipate that the majority of new applications will utilize a library of previously developed firmware modules similar to the common use of software numerical library routines for scientific computations. Early entries in the firmware module library include: exact match text search, approximate match text search, encryption, decryption, compression, decompression, etc. Clearly, the appropriate set of functional modules must be determined to some extent by experimentation and experience.

Given the availability of a library of functions developed for firmware, it becomes straightforward to provide a standard, published API that enables software developers to utilize the library effectively. Low-level details of driver interactions with the hardware interfaces can therefore be effectively hidden from (transparent to) the application software developer.

5. Unstructured Data Sets

The availability of inexpensive data storage has enabled the amassing of large amounts of information. At present, these data sets far exceed the capacity of modern processors, so searching them has become a serious challenge. In a recent invited talk [12] at the High Performance Embedded Computing Workshop, John Reynders of Celera Genomics commented that, “The size of the databases we deal with is no longer measured in terabytes, but in exabytes.”

Estimates are that in excess of 1.5 million web pages are added to the Internet each day. Companies that operate search engines have a difficult time keeping up with the torrent of information that requires indexing. Today, the performance bottleneck is the time taken to develop the required reverse index in reasonable time. Quoting from Domingos and Hulten [13]:

> In a single day WalMart records 20 million sales transactions, Google handles 70 million searches, and AT&T produces 275 million call records. Current algorithms for mining complex models from data (e.g., decision trees, sets of rules) cannot mine even a fraction of this data in useful time. Further, mining a day’s worth of data can take more than a day of CPU time, and so data accumulates faster than it can be mined.

Further complicating the situation is the fact that many data mining operations require searching a large unstructured space for approximate matches, and unstructured data that is rapidly changing is ill suited to reverse indexing. The two examples given above, genetics and the Internet, are illustrative of this. These same factors are also at the root of the problems associated with extracting useful intelligence from free form text data. The problem here is indeed even more difficult since the data tends to be very unstructured and the keys used in indexing are continuously evolving over time.

Approximate Keyword Search

Given the need to examine the entire data set, our architecture provides this capability with very high throughput via the FPGAs. For the approximate text search application illustrated in the last section, we have measured sustained data rates of 667 MB/s from disk to FPGA on a single compute node [4].

Figure 7 (from [5]) presents a comparison of the time required to search a data set given a relatively complex approximate match for the query (four keywords, each of length greater than 25 characters, and 10 character misses allowed in each keyword). Each data point represents an individual run, and the lines represent a linear curve fit to the data. The software execution is on an Intel Xeon 2.6 GHz machine with 2 GB of RAM, while the hardware execution is on the same platform with an FPGA added (connectivity provided via the PCI-X bus). The disk subsystem is a SCSI RAID deployed across a set of four Ultra320 SCSI channels.

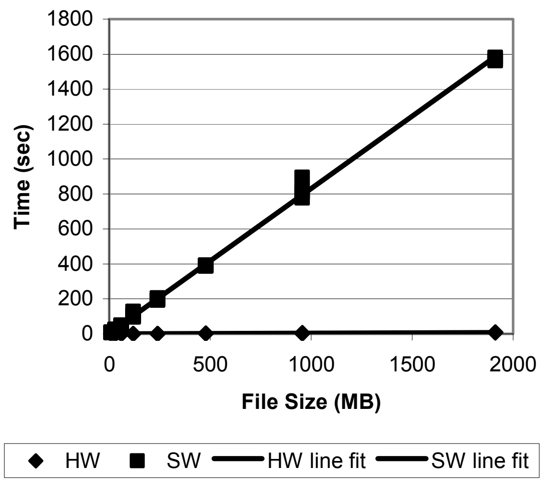

Figure 7. Performance comparison for approximate text search application [5].

The results are dramatic, with the FPGA-enabled system achieving a 200-fold performance gain over that of the software-only implementation.

The biosequence search application is quite similar to the above problem. Here, a query string (composed from the alphabet {A,C,G,T}) is searched across a large data set representing the known DNA sequence of one or more organisms. In this case, a more general approximate match (that is biologically significant) is desired. A classic approach to the solution to this problem is embodied in the BLAST family of software [14]. Based on a performance model calibrated with a partial implementation, a 100-fold performance gain over software BLAST is predicted on our system [3].

Semantically Informed Data Store

In many applications, a simple (even approximate) keyword search is not sufficient. Large volumes of data, whether they be commercial transaction records or unstructured text, are being generated continuously. While these data are being retained currently, the computational ability to usefully mine them is limited using current approaches.

Our system supports the development of a "semantically informed" data store. In traditional database systems, the structure of the database itself (its *schema*) serves to structure the kinds of queries that can be posed against the database. Formulation of a schema presupposes knowledge of the form of the acquired data as well as the kinds of queries likely to be posed.

We are interested in unstructured data, and the queries themselves will evolve according to the needs of the user community. As such, the traditional decomposition of a database into attributes is overly static and cannot suffice here.

Our system supports the development of dynamic schemas, whose attributes are not based necessarily on the direct contents of a data item, but are instead based on models of the data that serve to identify semantically relevant information within a data item. For unstructured text documents, these models come from computational linguistics (see, for example, [15]). A model essentially induces a dynamic schema on the documents, from which traditional projections, selections, and joins can be performed. Because the model is dynamic, it must be applied as the data is retrieved from the storage system. The architecture we describe is ideal for this, as illustrated below.

Figure 8 shows a data flow diagram appropriate for the semantically informed data store. The primary data are analyzed, forming semantic annotations that describe the data at a higher level of abstraction. These semantic annotations are retained in a separate data store. Due to the dynamic nature of the queries likely to be posed against the data, the semantic annotation store is used to guide decisions as to the primary data that requires re-analysis.

Both the semantic analysis functions and the re-analysis decision functions require the processing of large volumes of data. The ability to perform computations close to the data using reconfigurable hardware is of significant benefit to both of these needs.

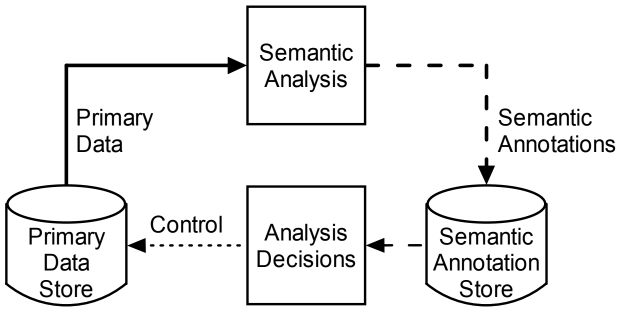

Figure 8. Data flow diagram.

The primary data store and the semantic annotation store are both (at their core) file systems that are retained on sets of coordinated disk drives. There are significant differences, however, in the details of how each data store is designed, and these differences can have a dramatic impact on enabling each subsystem to perform its assigned functions effectively. The primary data store is responsible for fast storage and retrieval of the original data. These data are the source material for the analysis and can be characterized as "write once, read many times" information. The subsection below describes a file system organization well suited to this class of data.

The semantic annotation store is responsible for retaining the results of data analysis. As such, it is subject to more balanced read/write requirements (e.g., data re-analysis will necessitate the alteration of preexisting semantic annotations present in the data store). A critical performance issue here is the relative data volume in the semantic annotations versus the primary data. If semantic annotation data volume approaches the size of the primary data, the semantic annotation store's throughput requirements will approach that of the primary data store.

Figure 9 illustrates how system components scale when increasing the capacity of the system through parallelism. The primary data analysis function is replicated several times. By partitioning the primary data, these functions can be reasonably executed in parallel, thereby increasing system capacity (both total storage and analysis throughput) linearly with the number of replicated units. For example, if each primary data store has a capacity of 10 TB and a 500 MB/s throughput, 100 units provide a total data capacity of 1 PB (10^{15} bytes) which, when one-half full, can be fully re-analyzed in approximately 2.5 hours.

This straightforward scaling fails, however, with the semantic annotation store. Figure 9 shows the semantic annotation information from the parallel analysis units all being forwarded to a single, global semantic annotation store. Re-analysis decisions made from the semantic annotation data are then fed back to the individual primary data stores to trigger re-analysis of primary data. A parallel, scalable, high-performance file system will be required to meet the needs of the semantic annotation store. In addition to the file system described below, another option is the Lustre file system [16], currently in use for large-scale scientific computing.

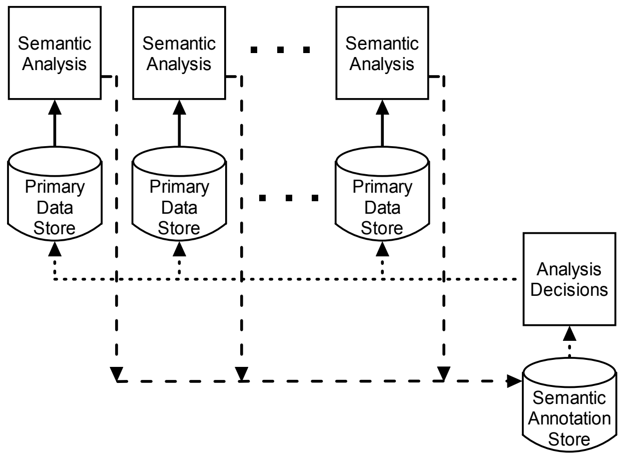

Figure 9. System scaling.

File System

Today's high-end disk storage systems are designed to deliver single files with high speed and reliability by leveraging the parallelism inherent in an array of disks. We have developed a new method of managing storage that, in addition to offering the parallelism of a disk array, also manages file layout within a disk so as to minimize the number of seeks (disk-head movements) required to deliver a file.

Our approach is based on Knuth's buddy system [17], in which a contiguous block of storage can be recursively subdivided into two blocks, each half the size of the original block. We have developed a new defragmentation algorithm [18] that can rearrange files on disk to allow a file to reside in (nearly) contiguous storage, dramatically reducing the number of seeks required to access a file. Extant file systems allocate a variable number of fixed-size blocks to a file, which can result in a number of seeks that is proportional to the file size. In our approach, at one extreme, a single block of appropriate size is allocated to a file, so that just a single seek is required. In between, our approach can allocate a number of blocks proportional to the log of the file's size, by allocating a sequence of blocks—each block being a power of 2 in size—whose cumulative size accommodates the file.

6. SUMMARY AND CONCLUSIONS

Magnetic storage capacities have grown faster than semiconductor features densities over the past decade. Bus speeds that interconnect the storage system to the traditional processor have had performance improvements at an even slower growth rate. The result has been the retention of vast quantities of data that we don't have the time to access or process.

We have built a system that addresses the above issues by moving the computation closer to the data and performing computation in reconfigurable hardware. The result is two orders of magnitude improvement in the overall computational throughput for high-volume data processing.

An important application that this system enables is the extraction of knowledge from unstructured data sets. The organization of the data must be gleaned from the data, and the semantic meaning (or importance) of the original data often changes with time. This implies significant, high-volume analysis and re-analysis of the full data set, something that is impractical using conventional approaches.

REFERENCES

[1] Roger D. Chamberlain, Ron K. Cytron, Mark A. Franklin, and Ronald S. Indeck, "The *Mercury* System: Exploiting Truly Fast Hardware for Data Search." In *Proc. of Int'l Workshop on Storage Network Architecture and Parallel I/Os,* September 2003, pp. 65-72.

[2] Qiong Zhang, Roger D. Chamberlain, Ronald S. Indeck, Benjamin West, and Jason White, "Massively Parallel Data Mining Using Reconfigurable Hardware: Approximate String Matching." In *Proc. of Workshop on Massively Parallel Processing,* April 2004.

[3] Praveen Krishnamurthy, Jeremy Buhler, Roger Chamberlain, Mark Franklin, Kwame Gyang, and Joseph Lancaster, "Biosequence Similarity Search on the Mercury System." In *Proc. of the IEEE 15th Int'l Conf. on Application-Specific Systems, Architectures and Processors*, September 2004, pp. 365-375.

[4] Roger Chamberlain, Berkley Shands, and Jason White, "Achieving Real Data Throughput for an FPGA Co-Processor on Commodity Server Platforms." In *Proc. of 1st Workshop on Building Block Engine Architectures for Computers and Networks*, October 2004.

[5] Mark Franklin, Roger Chamberlain, Michael Henrichs, Berkley Shands, and Jason White, "An Architecture for Fast Processing of Large Unstructured Data Sets." In *Proc. of 22nd Int'l Conf. on Computer Design*, October 2004, pp. 280-287.

[6] Gordon E. Moore, "Cramming More Components onto Integrated Circuits." *Electronics*, **38**(8), April 19, 1965.

[7] *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 1993 to 2004.

[8] *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*, 1993 to 2004.

[9] www.altera.com

[10] www.xilinx.com

[11] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software." *ACM Computing Surveys*, **34**(2):171-210, June 2002.

[12] John Reynders, "Computing Biology." Invited talk at 5th High Performance Embedded Computing Workshop, November 2001.

[13] Pedro Domingos and Geoff Hulten, “Catching Up with the Data: Research Issues in Mining Data Streams.” In *Proc. of Workshop on Research Issues in Data Mining and Knowledge Discovery*, May 2001.

[14] S. F. Altschul et al. “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.” *Nucleic Acids Research*, **25**:3389–402, 1997.

[15] J. Allanach, H. Tu, S. Singh, K. Pattipati and P. Willett, “Detecting, Tracking and Counteracting Terrorist Networks via Hidden Markov Models.” In *Proc. of IEEE Aerospace Conference*, March 2004.

[16] Cluster File Systems, “Lustre: A Scalable High-Performance File System.” White Paper, 2002.

[17] Donald E. Knuth, *The Art of Computer Programming*, volume 1, Addison-Wesley, 1973.

[18] Sharath Cholleti, *Storage Allocation in Bounded Time*, MS Thesis, Washington University, 2002.

Biography

Roger D. Chamberlain is director of engineering at Data Search Systems, Inc., and is an associate professor in the Department of Computer Science and Engineering at Washington University in St. Louis. His research areas include parallel processing, computer architecture, and reconfigurable systems. Dr. Chamberlain received the BSCS and BSEE degrees in 1983, the MSCS degree in 1985, and the DSc degree in computer science in 1989, all from Washington University.

Ron K. Cytron is a member of the technical staff at Data Search Systems, Inc., and is a professor in the Department of Computer Science and Engineering at Washington University in St. Louis. He previously served as Research Staff Member at IBM. His research areas include embedded systems, compilers, and real-time systems. Dr. Cytron received the BSEE degree in 1980 from Rice University and the MS degree in 1982 and PhD degree in 1984, both from the University of Illinois.